

# PA2 Document

Written by Hyeonho Shin, 20203344, Master Course, 010-5136-6003

## Summary

My homework is Resnet50-based, which is the well-known Net in image-classification area. But for better operating in our set I used more tricks (Adaptive learning rate, small batch size, modifying ResNet50[1], Kaiming initialization, and Data Augmentation)

The simplest way is to execute 'ForTA.py'. It loads the weights what I trained using the various technique.

File name	Role	Command
ForTA	Variation of TA code. Scoring program.	Python ForTA.py -i model_file_name.pth
tanukiChar_trainer	Training my Net, and returning model file.	Python tanukiChar_trainer.py
tanukiCharNet	Includes Network structure. I separated it for reducing redundancies.	None. Custom library
tanukiDataAug	Includes custom data augmentation class. In other words, preprocess programs are in it.	None. Custom library

## How to use

### For scoring

Execute ForTA.py. There is no need to use any arguments. It reads "FtanukiCharNet.pth" weight file on the same folder.

### Where are preprocessing and classifying functions?

All preprocessing steps are included in tanukiDataAug.py. Trainers call them before learning.

It follows Pytorch's transform class. Whenever you read batch data from disk, it preprocesses the images as many as batch size.

In summary, using tanukiChar\_trainer.py, what you need to do is...

1. Run tanukiChar\_trainer.py

2. Take the model having the best accuracy from step 2.
3. Run ForTA.py for evaluating. If you designate the input file name, type 'python ForTA.py - i input\_file.pth'

## Requisites

- Pytorch (1.7.0), (in 1.3.1 it will make error because save method is different.)
- Torchvision
- tanukiCharNet.py, tanukiDataAug.py in same folder.

## Design

### Abstract

For overcoming the small dataset, I adopted 4 techniques.

#### 1. Augmentation.

My codes generate randomly rotated, shifted, and resized figures without loss. In short, different to the common transform functions, the whole character is not destroyed. It can be achieved by boundary extraction. My code extracts the ROI (Region of Interest) using bounding box function. To make the function well-operated, I applied bilateralFilter and adaptive thresholding in prior. And then for denoising, I applied randomly contrast increasing. From ROI, I resized ROI randomly, but using LANCZOS for lower resizing noise. And then translate it in only 350\*350 array.

Because of the speed problem, I implemented this by two methods. First one is simple, I made custom transform function for Pytorch. It looks nice and easy to use. (It is implemented in *tanukiChar\_trainer.py*) But data loading is too slow because NumPy cannot support multiprocessing. For solving it, I just implemented the program to save augmented image in HDD first, "*gen\_aug.py*". And then train it with "*tanukiChar\_trainer\_with\_gen\_aug.py*". It is faster.

#### 2. Small batch size

By some papers, they argued that generalization performance increase in more various learning rate under small batch size. It recommends batch size as 8.

#### 3. Adaptive learning rate

At higher learning rate, we can get more faster learning. But there can be divergence or vibration. So I decrease the learning rate epoch by epoch.

#### 4. Kaiming initialization

For faster converging, I initialized my net using 'Kaiming initializer'. It has more better speed then one without that because the my net consists of 'ReLU'.

## 5. Modify ResNet50

Sadly, ResNet50 only supports 224\*224 images with 1000 classes. So I adopted AdaptiveAvgPool2d layer and modified the final FC layer to 5 without downscaling of input images.

## Results

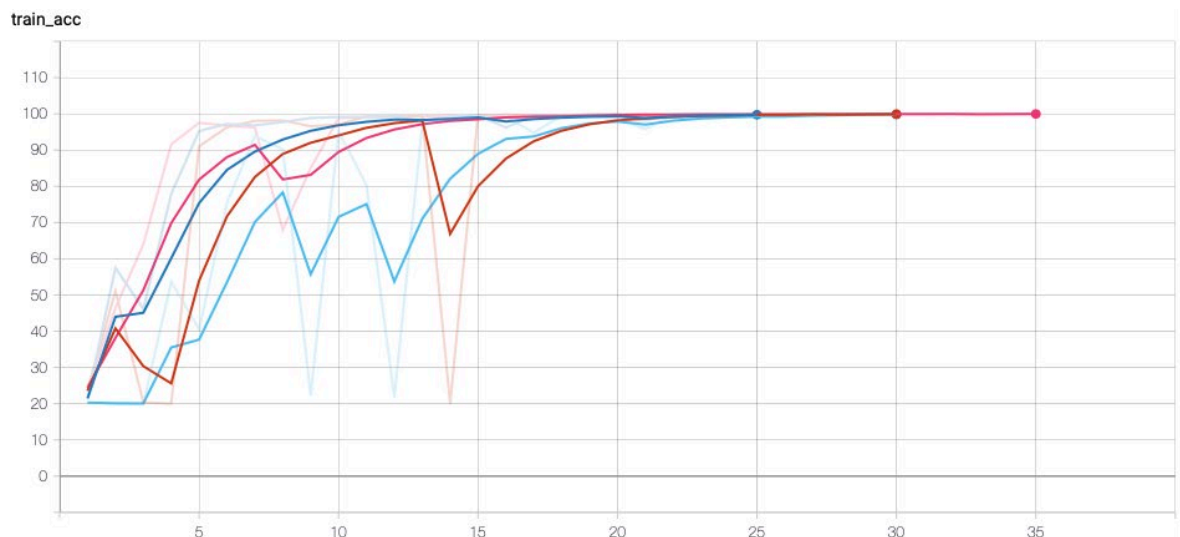
### Summary

I chose my model with epochs 33, batch size 8, learning rate 0.002. It was the best result in searching space batch size: 8 or 16, learning rate 0.001 or 0.002

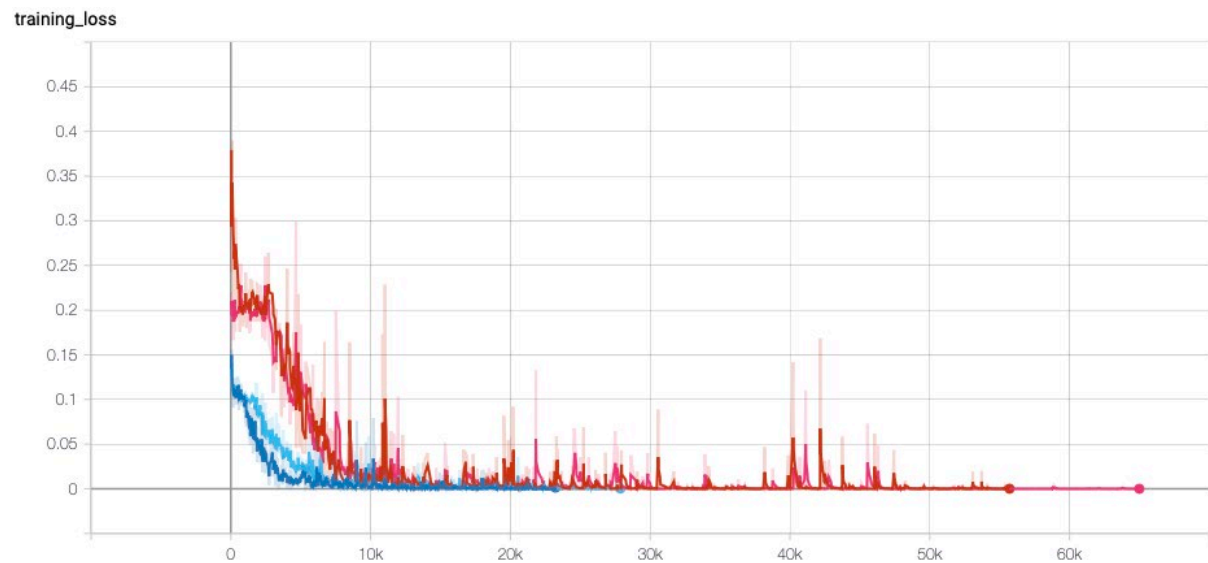
Gray lines are real value, and strong lines are poly-fitted value.

Batch size\Learning rate	0.001	0.002
8	Red	Claret(자홍)
16	Blue	Sky blue

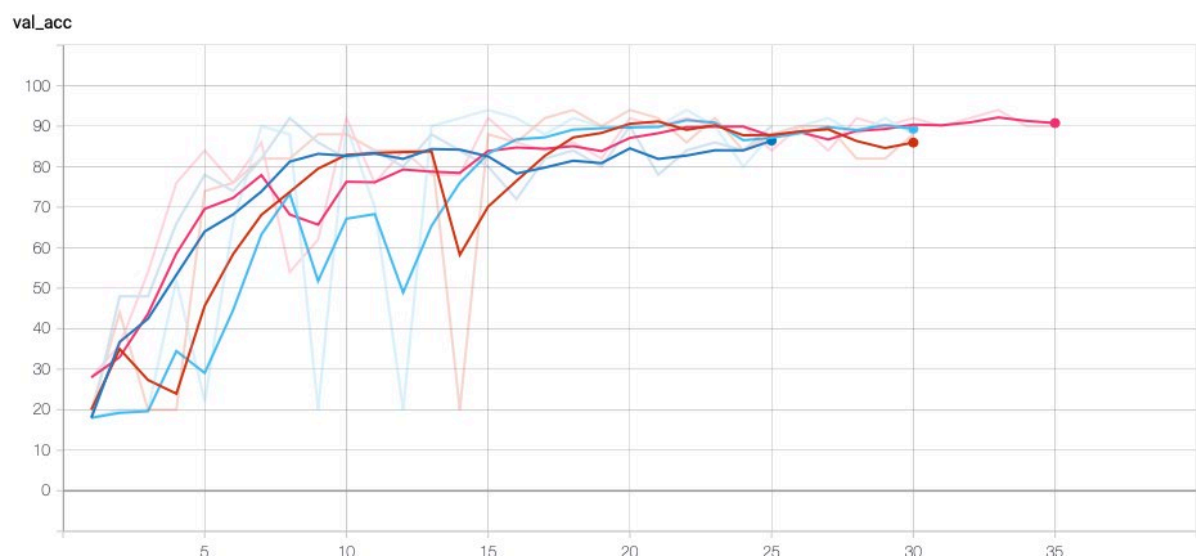
### Training Accuracy (by epochs)



## Training loss (by iterations)



## Validation Accuracy (by epochs)



## For faster learning

For faster learning, I separated Preprocessing function as 'gen\_aug.py' when training in real. It enhances, take ROI of images and save the results on your disk, './aug\_abcde'. So, in this process, all you need is

1. Run gen\_aug.py to preprocess.
2. Run tanukiChar\_trainer\_with\_gen\_aug.py.
3. Take the model having the best accuracy from step 2.
4. Run ForTA.py for evaluating. If you designate the input file name, type 'python ForTA.py - i input\_file.pth'

I omitted the files because you can be confused of many files. But you can see it at the my git( <https://github.com/hyeonhoshin/tanukiCharacters> )

## REFERENCES

[1] Pytorch's Official ResNet implementation, "ResNet50",  
<https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py>

[2] Pytorch's Official Tutorial, "모델 저장하기 & 불러오기",  
[https://tutorials.pytorch.kr/beginner/saving\\_loading\\_models.html](https://tutorials.pytorch.kr/beginner/saving_loading_models.html)

[3] Pytorch's Official Tutorial, "분류기(Classifier) 학습하기",  
[https://tutorials.pytorch.kr/beginner/blitz/cifar10\\_tutorial.html](https://tutorials.pytorch.kr/beginner/blitz/cifar10_tutorial.html)